UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/767,480	01/28/2004	Norman Rubin	00100.03.0040	5073

29153 7590 09/25/2007

ADVANCED MICRO DEVICES, INC.
C/O VEDDER PRICE KAUFMAN & KAMMHOLZ, P.C.
222 N.LASALLE STREET
CHICAGO, IL 60601

EXAMINER	
WANG, BEN C	

ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
09/25/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

MN

Office Action Summary	Application No.	Applicant(s)	
	10/767,480	RUBIN ET AL.	
	Examiner	Art Unit	
	Ben C. Wang	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *09 July 2007*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-9,11-16 and 18-24* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-5,8,9,11,14-16,18-19 and 21-24* is/are rejected.

7)☒ Claim(s) *6,7,12,13 and 20* is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *09 July 2007* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *7/9/2007*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Applicant's amendment dated July 9, 2007, responding to the Office action mailed February 9, 2007 provided in the rejection of claims 1-20, wherein claims 1, 4-5, 8-9, 11, 14-16, and 18-19 are amended, claims 10 and 17 are canceled, and claims 21-24 are new.

Claims 1-9, 11-16, and 18-24 remain pending in the application and which have been fully considered by the examiner.

Applicant's arguments with respect to claims rejection have been fully considered but are moot in view of the new grounds of rejection – see *Cytron et al.* - art made of record, as applied hereto.

Claim Rejections – 35 USC § 112

The following is a quotation of the second paragraph of 35 U.S.C. 112:
The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

2. Claims 1-7, 13-21, and 23-24 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

3. **As to claim 1**, "the machine code" is recited, line 11; however, there is no "a machine code" previously described within the claim; the claim lacks proper antecedent basis and which it has been treated as such "a machine code" by the Examiner.

4. **As to claims 2-7**, they are rejected as including the deficiency in the independent claim 1.

5. **As to claim 7** and **claim 13**, the term "may be written" is not a positive recitation term which renders the claim indefinite. The term "may be written" is not defined by the claim, the specification does not provide a standard for ascertaining the requisite degree, and one of ordinary skill in the art would not be reasonably apprised of the scope of the invention.

6. **As to claim 14**, "the machine code" is recited, line 9; however, there is no "a machine code" previously described within the claim or the base claim; the claim lacks proper antecedent basis and which it has been treated as such "a machine code" by the Examiner.

7. **As to claims 15-18, 21, and 23**, they are rejected as including the deficiency in the independent claim 14.

8. **As to claim 18**, "the superword register" is recited, line 3; however, there is no "a superword register" previously described within the claim or the base claim; the claim lacks proper antecedent basis and which it has been treated as such "a superword register" by the Examiner.

9. **As to claim 19**, "the machine code" is recited, line 16; however, there is no "a machine code" previously described within the claim; the claim lacks proper antecedent basis and which it has been treated as such "a machine code" by the Examiner.

10. **As to claims 20, and 23**, they are rejected as including the deficiency in the independent claim 19.

Claim Rejections – 35 USC § 102(b)

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102(b) that form the basis for the rejections under this section made in this office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

11. Claims 1-5, 8-9, 11, 14-16, 19, and 21-24 are rejected under 35 U.S.C. 102(b) as being anticipated by Cytron et al., (*Efficiently Computing Static Single Assignment Form and the Control Dependence Graph, 1991, ACM*) (hereinafter 'Cytron' - art made of record)

12. **As to claim 1**, Cytron discloses (currently amended) a method for static single assignment form dead code elimination, the method comprising:

- examining a first instruction off of a worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered) in memory, wherein the first instruction includes a previous link (e.g., Sec. 6 – Construction of Control Dependences, 1st Par. – we show that control dependences are essentially the dominance frontiers in the reverse graph of the control flow graph; P. 477, 2nd Par. – the reverse control flow graph RCFG has the same nodes as the control flow graph CFG, but has an edge Y → X for each edge X → Y in CFG; the roles of Entry and Exit are also reverse; Sec. 9.3 – Conclusions – applying the early steps in the SSA translation to the reverse graph is an efficient way to compute control dependences) and a write mask (e.g., P. 480, 1st Par., 1st point – Live(S) indicates that state S is live);
- examining at least one second instruction, wherein the at least one second instruction is a source of the first instruction and wherein each of the at least one second instruction includes a previous link and a write mask; determining if any components within a particular field of the at least one second instruction are required (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional

branch, and there are statements already marked live that are control dependent on this conditional branch);

- when no components of the at least one second instruction are required, deleting the at least one second instruction from the (a) machine code (e.g., Fig. 17 – Dead code elimination; P. 479, last Par. – our algorithm, given in Figure 17, goes one step further in eliminating dead conditional branches); and
- when any component of the at least one second instruction is required, adding the at least one second instruction to the worklist in the memory (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered).

13. **As to claim 2** (incorporating the rejection in claim 1) (original), Cytron discloses the method further comprising: generating the worklist by:

- for each of a plurality of instructions, determining if the instruction is a critical instruction (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch); and

- if the instruction is a critical instruction, writing the instruction to the worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered).

14. **As to claim 3** (incorporating the rejection in claim 2) (original), Cytron discloses the method further comprising: setting a live bit for each component of the plurality of instructions (e.g., Sec. 7.1 – Dead Code Elimination, 2nd Par.; P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch).

15. **As to claim 4** (incorporating the rejection in claim 2) (currently amended), Cytron discloses the method wherein each critical instruction is an instruction that generates an export value (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch).

16. **As to claim 5** (incorporating the rejection in claim 2) (currently amended), Cytron discloses the method further comprising: prior to generating the worklist: receiving a plurality of instructions; adding to each instruction the previous link (e.g., Sec. 6 – Construction of Control Dependences, 1st Par. – we show that control dependences are essentially the dominance frontiers in the reverse graph of the control flow graph; P. 477, 2nd Par. – the reverse control flow graph RCFG has the same nodes as the control flow graph CFG, but has an edge Y → X for each edge X → Y in CFG; the roles of Entry and Exit are also reverse; Sec. 9.3 – Conclusions – applying the early steps in the SSA translation to the reverse graph is an efficient way to compute control dependences); and adding to each instruction the write mask (e.g., P. 480, 1st Par., 1st point – Live(S) indicates that state S is live).

17. **As to claim 8** (currently amended), Cytron discloses a method for static single assignment form dead code elimination comprising:

- receiving a plurality of instructions; adding to each instruction a previous link (e.g., Sec. 6 – Construction of Control Dependences, 1st Par. – we show that control dependences are essentially the dominance frontiers in the reverse graph of the control flow graph; P. 477, 2nd Par. – the reverse control flow graph RCFG has the same nodes as the control flow graph CFG, but has an edge Y → X for each edge X → Y in CFG; the roles of Entry and Exit are also reverse; Sec. 9.3 – Conclusions –

applying the early steps in the SSA translation to the reverse graph is an efficient way to compute control dependences);

- adding to each instruction a write mask (e.g., P. 480, 1st Par., 1st point – Live(S) indicates that state S is live); and

generating a worklist in memory:

- for each of the plurality of instructions, determining if the instruction is a critical instruction (e.g.; P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch); and
- if the instruction is a critical instruction, writing the instructions to the worklist in the memory (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered).

18. **As to claim 9** (incorporating the rejection in claim 8) (currently amended), Cytron discloses the method of claim 8 further comprising: setting a live bit for each component of the plurality of instructions:

- examining a first instruction off of the worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered);

- examining at least one second instruction in the machine code, wherein the at least one instruction is a source of the first instruction (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch);
- determining if any component within a particular field of the at least one second instruction is live (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch); and
- when no components of the at least one second instruction are live, deleting the second instruction from the worklist (e.g., Fig. 17 – Dead code elimination; P. 479, last Par. – our algorithm, given in Figure 17, goes one step further in eliminating dead conditional branches).

19. **As to claim 11** (incorporating the rejection in claim 8) (currently amended), Cytron discloses the method wherein each critical instruction is an instruction that generates an export value (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch).

20. **As to claim 14** (currently amended), Cytron discloses an apparatus for static single assignment form dead code elimination comprising: at least one memory device storing a plurality of executable instructions; and at least one processor operably coupled to the at least one memory device, operative to receive the plurality of executable instructions such that the at least one processor, in response to plurality of executable instructions is further operative to:

- examine a first instruction off of a worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered), wherein the first instruction includes previous link (e.g., Sec. 6 – Construction of Control Dependences, 1st Par. – we show that control dependences are essentially the dominance frontiers in the reverse graph of the control flow graph; P. 477, 2nd Par. – the reverse control flow graph RCFG has the same nodes as

the control flow graph CFG, but has an edge Y → X for each edge X → Y in CFG; the roles of Entry and Exit are also reverse; Sec. 9.3 – Conclusions – applying the early steps in the SSA translation to the reverse graph is an efficient way to compute control dependences) and a write mask (e.g., P. 480, 1st Par., 1st point – Live(S) indicates that state S is live);

- examine at least one second instruction of the (a) machine code, wherein the at least one second instruction is a source of the first instruction and each of the at least one second instruction includes a previous link and a write mask; determine if any component within a particular field of the at least one second instruction is live (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch); and
- when no components are live, delete the second instruction from the machine code (e.g., Fig. 17 – Dead code elimination; P. 479, last Par. – our algorithm, given in Figure 17, goes one step further in eliminating dead conditional branches).

21. **As to claim 15** (incorporating the rejection in claim 14) (currently amended), Cytron discloses the apparatus wherein the at least one processor in response to the plurality of instructions executable instructions is further operative to: generate the worklist by:

- for each of a plurality of instructions, determining if the instruction is a critical instruction (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch); and
- if the instruction is a critical instruction, writing the instruction to the worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered).

22. **As to claim 16** (incorporating the rejection in claim 15) (currently amended), Cytron discloses the apparatus wherein critical instruction is an instruction that generates an export value (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch,

and there are statements already marked live that are control dependent on this conditional branch).

23. **As to claim 19** (currently amended), Cytron discloses an apparatus for static single assignment form dead code eliminations comprising: at least one memory device storing a plurality of executable instructions; and at least one processor operably coupled to the at least one memory device, operative to receive the plurality of executable instructions such that the at least one processor, in response to the executable instructions is further operative to:

- receive a plurality of instructions; add to each instruction a previous link (e.g., Sec. 6 – Construction of Control Dependences, 1st Par. – we show that control dependences are essentially the dominance frontiers in the reverse graph of the control flow graph; P. 477, 2nd Par. – the reverse control flow graph RCFG has the same nodes as the control flow graph CFG, but has an edge Y → X for each edge X → Y in CFG; the roles of Entry and Exit are also reverse; Sec. 9.3 – Conclusions – applying the early steps in the SSA translation to the reverse graph is an efficient way to compute control dependences);
- add to each instruction a write mask (e.g., P. 480, 1st Par., 1st point – Live(S) indicates that state S is live); and
- generate a worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered) by:

- for each of the plurality of instructions; determining if the instruction is a critical instruction (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch); and
- if the instruction is a critical instruction, writing the instructions to the worklist:
- examine a first instruction off of the worklist;
- examine at least one second instruction from the machine code, wherein the at least one second instruction is a source of the first instruction (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch);
- determine if any component within a particular field of the at least one second instruction is live; and when no component is live, delete the second instruction from the machine code (e.g., Fig. 17 – Dead code elimination; P. 479, last Par. – our

algorithm, given in Figure 17, goes one step further in eliminating dead conditional branches).

24. **As to claim 21** (incorporating the rejection in claim 15) (new), Cytron discloses the apparatus wherein the at least one processor in response to the plurality of executable instructions is further operative to set a live bit for each component of the plurality of instructions (e.g., Sec. 7.1 – Dead Code Elimination, 2nd Par.; P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch).

25. **As to claim 22** (incorporating the rejection in claim 9) (new), Cytron discloses the method further comprising: when any component of the at least one second instruction is live (e.g., Sec. 7.1 – Dead Code Elimination, 2nd Par.; P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control

dependent on this conditional branch), adding the at least one second instruction to the worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered).

26. **As to claim 23** (incorporating the rejection in claim 14) (new), Cytron discloses the apparatus wherein the at least one processor in response to the plurality of instructions executable instructions is further operative to: when any component of the at least one second instruction is live (e.g., Sec. 7.1 – Dead Code Elimination, 2nd Par.; P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch), add the at least one second instruction to the worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered)t.

27. **As to claim 24** (incorporating the rejection in claim 19) (new), Cytron discloses the apparatus wherein the at least one processor in response to the plurality of instructions executable instructions is further operative to: when any component of the at least one second instruction is live (e.g., Sec. 7.1 – Dead Code Elimination, 2nd Par.; P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a

reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch)., add the at least one second instruction to the worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered).

Allowable Subject Matter

28. Claims 6-7, 12-13, and 20 are objected to as being dependent upon a rejected base claim, but would be allowable if rewritten to overcome all the limitations of the base claim and any intervening claims.

The following is an examiner's statement of reasons for allowance:

29. **Regarding claims** 6-7, 12-13, and 20, prior art of record fails to reasonably show or suggest "the write mask is a multi-bit field representing a number of components in a superwork register; each of the plurality of instructions (may be) are written to the worklist a predetermined number of times, wherein the predetermined number of times is based on the number of components in the superword register".

Conclusion

30. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-

1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

TUAN DAM
SUPERVISORY PATENT EXAMINER

BCW

September 12, 2007